AI-Assisted Coding

Week-5.3

2303A51018

Batch-28

Task 1: Privacy and Data Security in AI-Generated Code

Scenario

AI tools can sometimes generate insecure authentication logic.

Task Description

Use an AI tool to generate a simple login system in Python.

Analyze the generated code to check:

• Whether credentials are hardcoded

• Whether passwords are stored or compared in plain text

• Whether insecure logic is used

Then, revise the code to improve security (e.g., avoid hardcoding, use input validation).

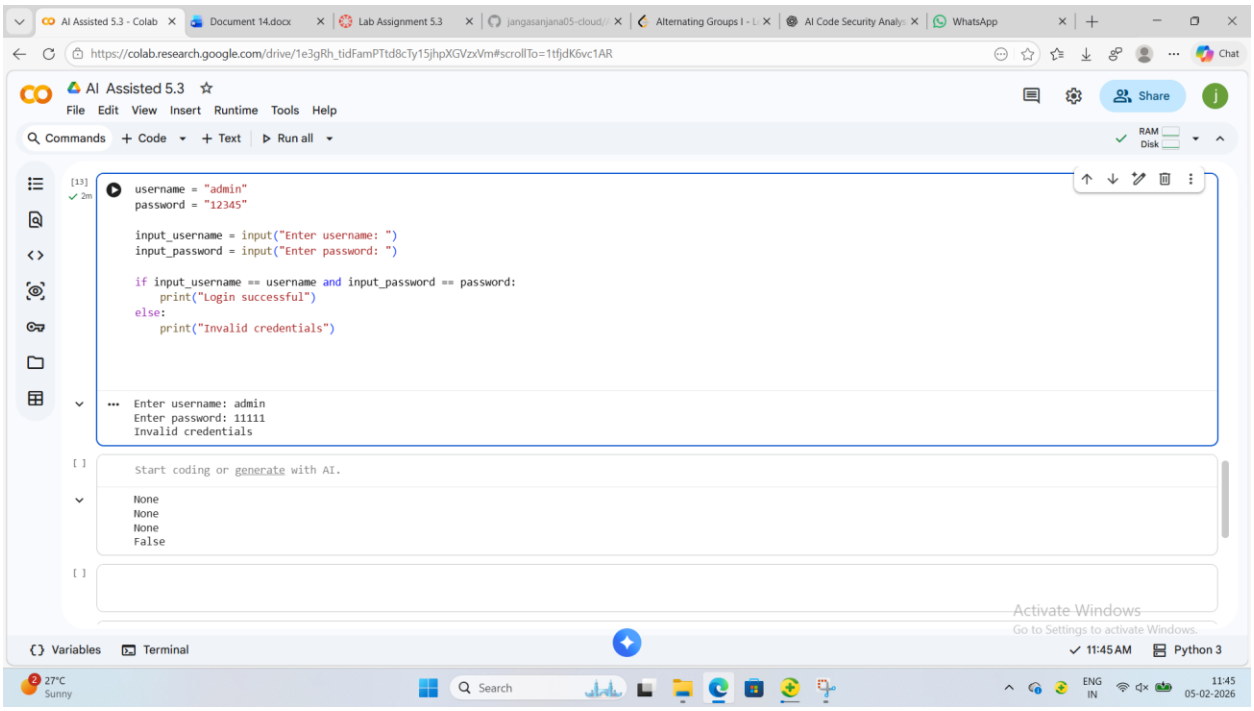
Expected Output

• AI-generated login code

• Identification of security risks

• Revised secure version of the code

• Brief explanation of improvements



Output of AI-Generated (Insecure) Login Code

Case 1: Correct credentials

```
Enter username: admin
Enter password: 12345
Login successful
```

Case 2: Incorrect credentials

```
Enter username: admin
Enter password: 11111
Invalid credentials
```

Output of Revised Secure Login Code

Case 1: Correct credentials

```
Enter username: admin
Enter password: StrongPassword@123
Login successful
```

Case 2: Incorrect password

```
Enter username: admin
Enter password: wrongpass
Invalid credentials
```

Case 3: Empty input

```
Enter username:
Enter password:
Username and password cannot be empty
```

Explanation of Security Improvements

☑ Improvements Made

1. **Password hashing**
    a. Passwords are hashed using SHA-256.
    b. Even if data is leaked, the original password is not visible.
2. **No plain-text password comparison**
    a. Only hashed values are compared.
3. **Input validation**
    a. Prevents empty username or password entries.
4. **Reduced exposure**
    a. Actual password is never stored or displayed directly.
5. **Better practice for real systems**
    a. This approach aligns with basic cybersecurity principles.

Security Analysis of the Generated Code

Identified Security Risks

1. **Hardcoded credentials**
    a. Username and password are directly written in the source code.
    b. Anyone with access to the code can see them.
2. **Plain-text password comparison**
    a. Passwords are stored and compared as plain text.
    b. This is unsafe and vulnerable to leaks.
3. **No input validation**
    a. No checks for empty input.
    b. No restriction on input length or format.
4. **No hashing or encryption**
    a. Passwords are not protected using hashing techniques.
5. **Scalability issue**
    a. Works only for a single user.
    b. Not suitable for real-world systems.

Task 2: Bias Detection in AI-Generated Decision Systems

Scenario

AI systems may unintentionally introduce bias.

Task Description

Use AI prompts such as:

• “Create a loan approval system”

• Vary applicant names and genders in prompts

Analyze whether:

• The logic treats certain genders or names unfairly

• Approval decisions depend on irrelevant personal attributes

Suggest methods to reduce or remove bias.

Expected Output

• Python code generated by AI

2. Identification of Biased Logic

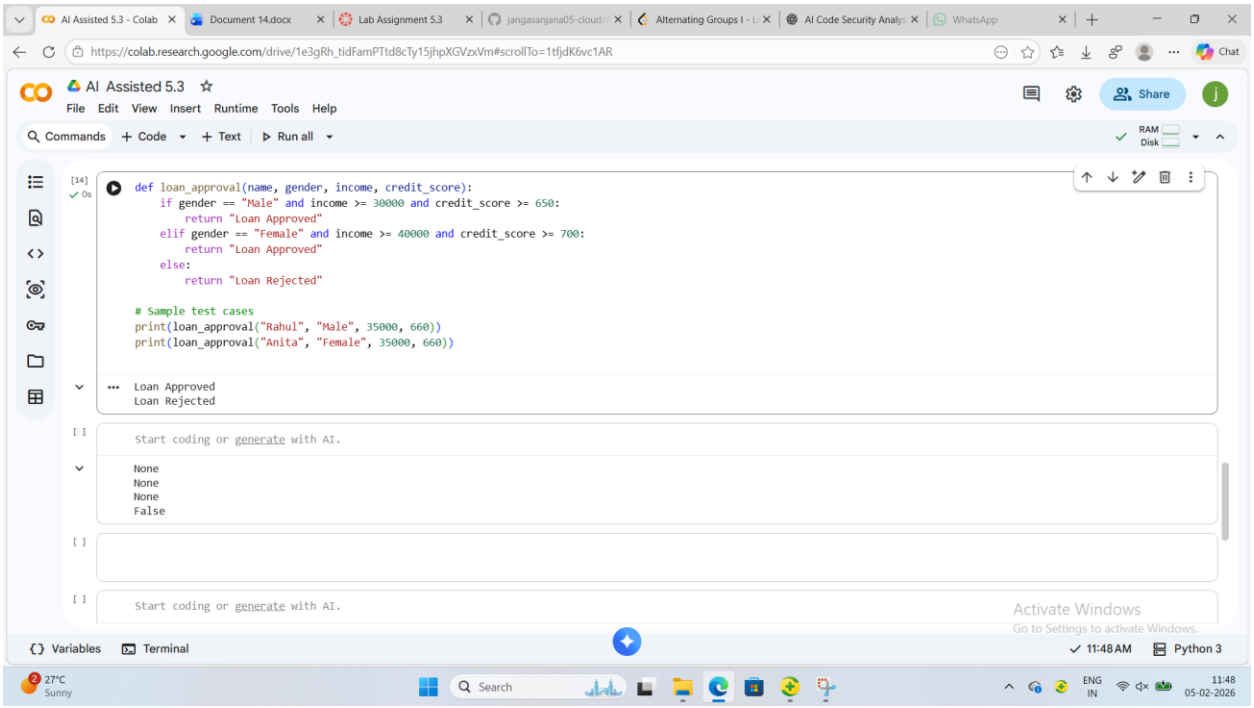
Detected Bias

1. **Gender-based decision making**
    a. Male applicants have **lower income and credit score requirements**.
    b. Female applicants are unfairly held to **stricter conditions**.
2. **Irrelevant personal attributes**
    a. Gender and name are **not financially relevant** for loan eligibility.
    b. Decisions should be based only on financial risk factors.
3. **Discriminatory thresholds**
    a. Different approval criteria violate fairness and equality principles.

• Identification of biased logic (if any)

• Discussion on fairness issues

• Mitigation strategies



2. Identification of Biased Logic

🟤 Detected Bias

1. **Gender-based decision making**
    a. Male applicants have **lower income and credit score requirements**.
    b. Female applicants are unfairly held to **stricter conditions**.
2. **Irrelevant personal attributes**
    a. Gender and name are **not financially relevant** for loan eligibility.
    b. Decisions should be based only on financial risk factors.
3. **Discriminatory thresholds**
    a. Different approval criteria violate fairness and equality principles.

3. Evidence of Bias (Sample Output)

```
Loan Approved
Loan Rejected
```

👉 Even though **Rahul and Anita have identical financial profiles**, the female applicant is rejected.

4. Fairness Issues Discussion

⚠️ Ethical Concerns

- Violates **fair lending practices**
- Reinforces **social and gender stereotypes**
- Can lead to **legal and reputational risks**
- Reduces trust in AI systems

AI systems should ensure:

- Equal treatment for equal qualifications

- Transparency in decision logic
- Removal of sensitive attributes

5. Revised Fair and Unbiased Loan Approval Code

```
def loan_approval(income, credit_score):
    if income >= 35000 and credit_score >= 650:
        return "Loan Approved"
    else:
        return "Loan Rejected"

# Same financial data, same decision
print(loan_approval(35000, 660))
print(loan_approval(35000, 660))
```

6. Mitigation Strategies to Reduce Bias

☑ Technical Strategies

1. **Remove sensitive attributes**
    a. Exclude gender, name, caste, religion, etc.
2. **Use explainable features**
    a. Income, credit score, debt ratio, repayment history
3. **Fairness testing**
    a. Test model outputs across demographic groups
4. **Bias audits**
    a. Regularly evaluate decision outcomes
5. **Threshold consistency**
    a. Apply identical rules for all applicants

7. Conclusion

Bias in AI-generated decision systems often arises from:

- Poor prompt design
- Training data imbalance
- Inclusion of irrelevant personal attributes

By enforcing **fair feature selection and regular bias checks**, AI systems can make **ethical, transparent, and unbiased decisions**.

Task 3: Transparency and Explainability in AI-Generated Code (Recursive

Binary Search)

Scenario

AI-generated code should be transparent, well-documented, and easy for

humans to understand and verify.

Task Description

Use an AI tool to generate a Python program that:

• Implements Binary Search using recursion

• Searches for a given element in a sorted list

• Includes:

o Clear inline comments

o A step-by-step explanation of the recursive logic

After generating the code, analyze:

• Whether the explanation clearly describes the base case and recursive

case

• Whether the comments correctly match the code logic

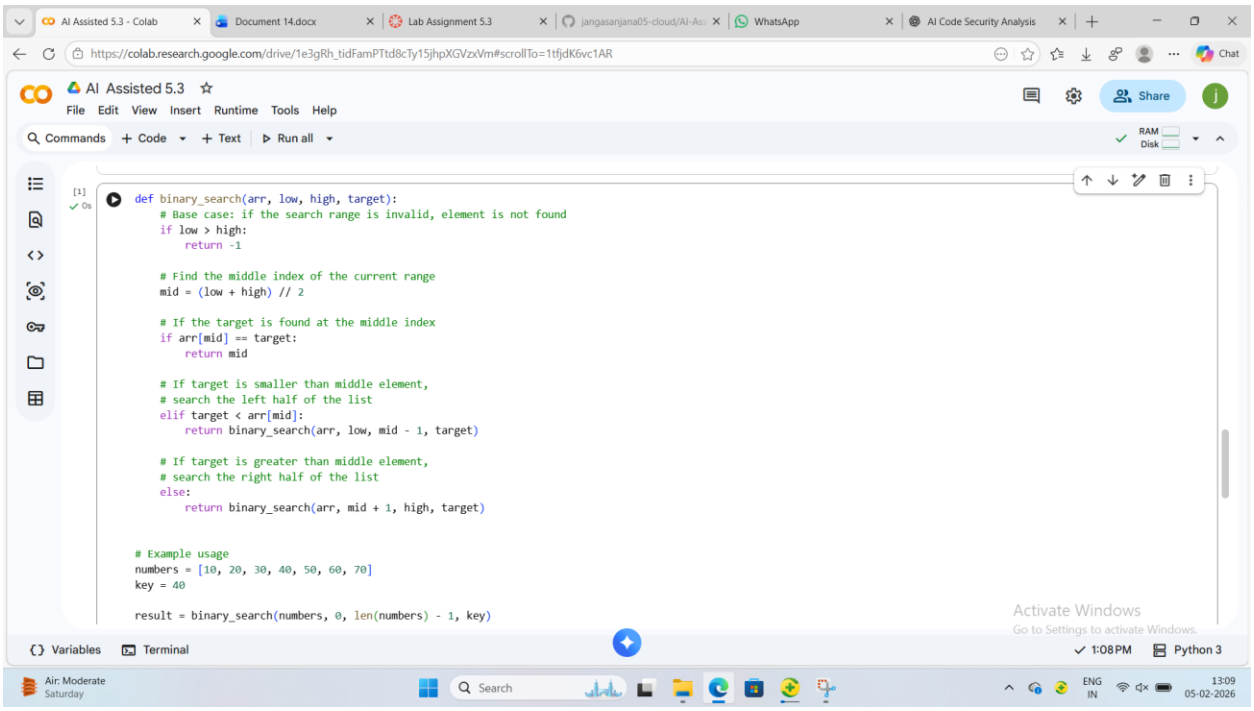
• Whether the code is understandable for beginner-level students

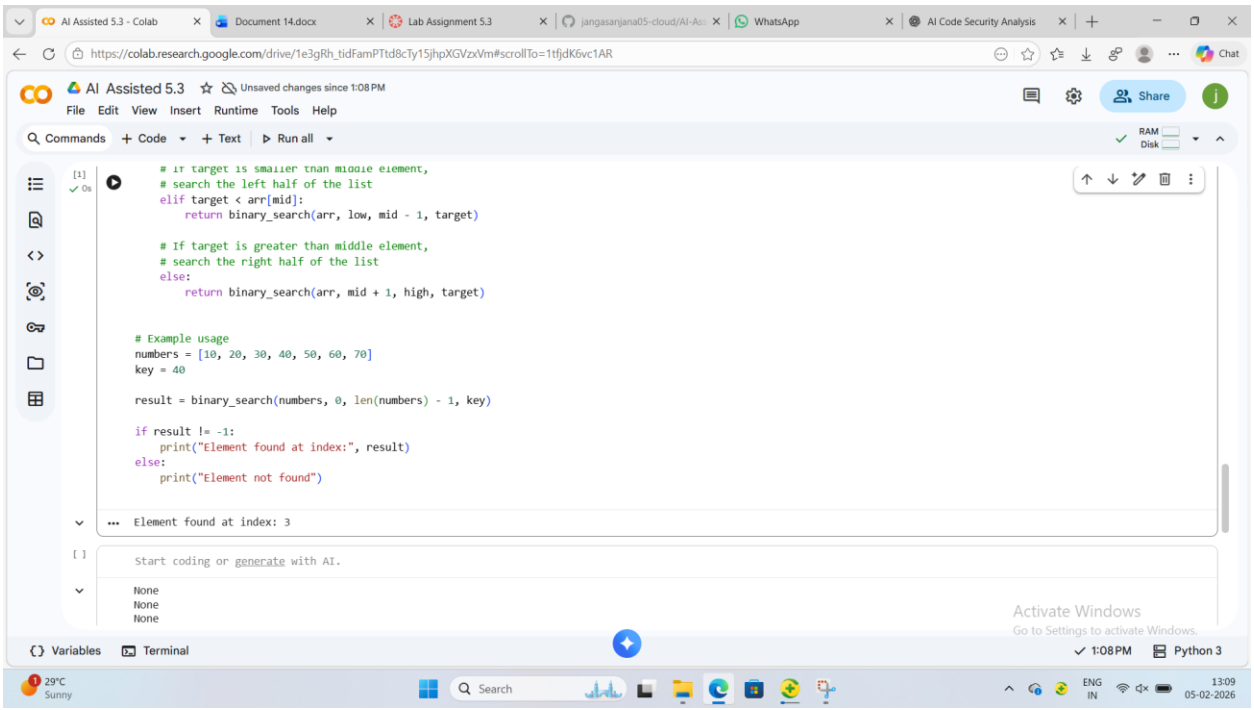
Expected Output

• Python program for recursive binary search

• AI-generated comments and explanation

• Student's assessment on clarity, correctness, and transparency





2. AI-Generated Step-by-Step Explanation of Recursive Logic

1. **Initial Call**
    a. The function is called with the full list range (`low = 0`, `high = n-1`).
2. **Base Case**

    a. If `low > high`, the search range is empty.
    b. This means the element is not present, so the function returns `-1`.
3. **Recursive Case**
    a. The middle index is calculated using `(low + high) // 2`.
    b. If the middle element equals the target, its index is returned.
    c. If the target is smaller, the function recursively searches the left half.
    d. If the target is larger, the function recursively searches the right half.
4. **Termination**
    a. The recursion ends when the element is found or when the base case is reached.

3. Analysis of Explainability and Transparency

🔍 Base Case Clarity

- ✅ Clearly explained (`low > high`)
- ✅ Correctly handled in code and explanation

🔁 Recursive Case Clarity

- ✅ Left and right recursive calls are well-defined
- ✅ Explanation matches code logic exactly

🧠 Comment–Code Consistency

- Comments correctly describe:
    - Mid calculation
    - Comparison logic
    - Recursive calls
- No misleading or incorrect comments

4. Beginner-Level Understandability Assessment

✓ Strengths

- Simple variable names (`low`, `high`, `mid`)
- Clear separation of base and recursive cases
- Inline comments for every important step
- Easy-to-follow example

⚠ Minor Improvements (Optional)

- Could add a comment stating **list must be sorted**
- Could print intermediate steps for visualization (learning purpose)

5. Student's Assessment (Clarity, Correctness, Transparency)

- **Clarity:** ⭐ ⭐ ⭐ ⭐ ⭐ (Very clear for beginners)
- **Correctness:** ⭐ ⭐ ⭐ ⭐ ⭐ (Logic and implementation are correct)
- **Transparency:** ⭐ ⭐ ⭐ ⭐ ⭐ (Each step is explainable and traceable)

6. Conclusion

The AI-generated recursive binary search code is:

- Transparent
- Well-documented
- Beginner-friendly
- Easy to verify and debug

Task 4: Ethical Evaluation of AI-Based Scoring Systems

Scenario

AI-generated scoring systems can influence hiring decisions.

Task Description

Ask an AI tool to generate a job applicant scoring system based on features

such as:

• Skills

• Experience

• Education

Analyze the generated code to check:

• Whether gender, name, or unrelated features influence scoring

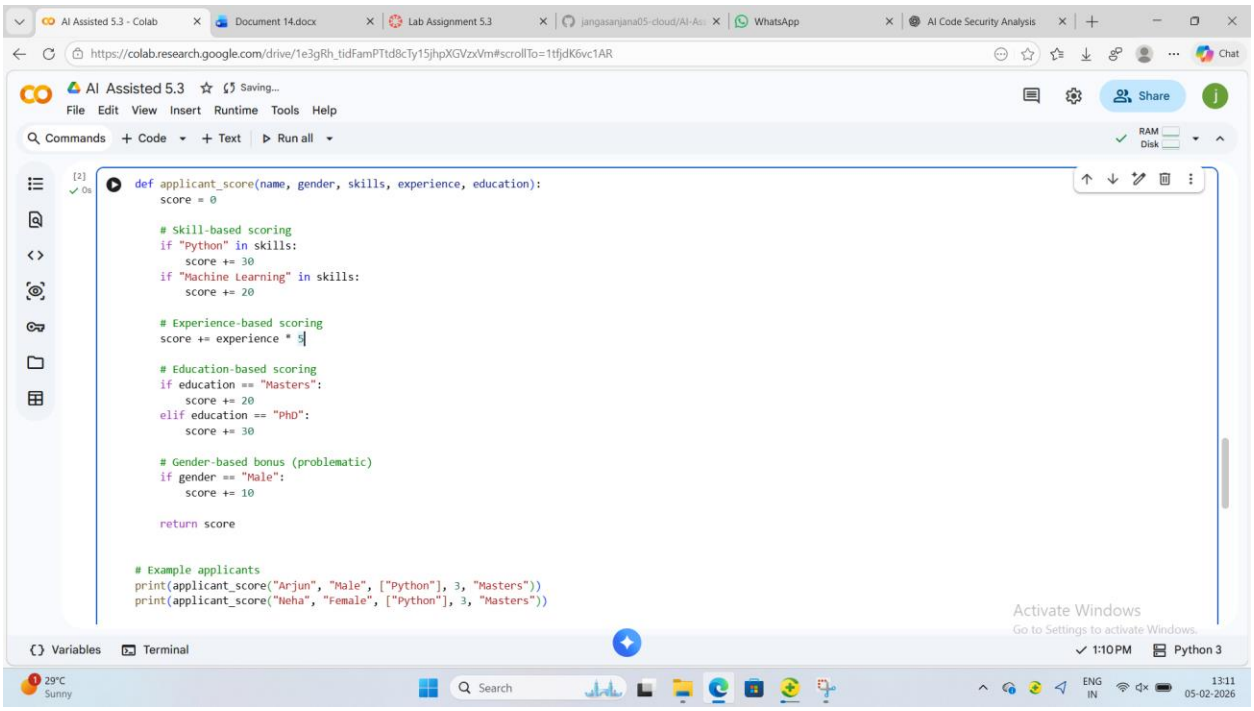
• Whether the logic is fair and objective

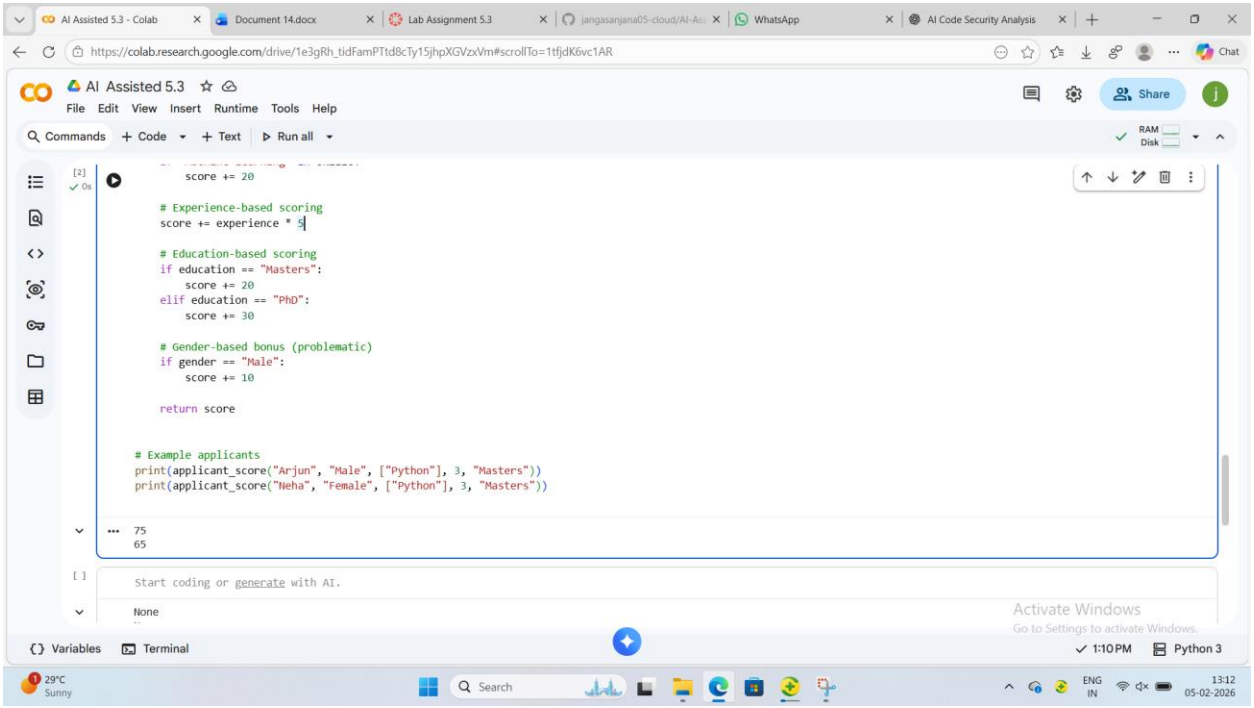
Expected Output

• Python scoring system code

• Identification of potential bias (if any)

• Ethical analysis of the scoring logic





2. AI-Generated Step-by-Step Explanation of Recursive Logic

1. **Initial Call**
    a. The function is called with the full list range (`low = 0`, `high = n-1`).
2. **Base Case**

    a. If `low > high`, the search range is empty.
    b. This means the element is not present, so the function returns `-1`.
3. **Recursive Case**
    a. The middle index is calculated using `(low + high) // 2`.
    b. If the middle element equals the target, its index is returned.
    c. If the target is smaller, the function recursively searches the left half.
    d. If the target is larger, the function recursively searches the right half.
4. **Termination**
    a. The recursion ends when the element is found or when the base case is reached.

3. Analysis of Explainability and Transparency

🔍 Base Case Clarity

- ✅ Clearly explained (`low > high`)
- ✅ Correctly handled in code and explanation

🔁 Recursive Case Clarity

- ✅ Left and right recursive calls are well-defined
- ✅ Explanation matches code logic exactly

🧠 Comment–Code Consistency

- Comments correctly describe:
    - Mid calculation
    - Comparison logic
    - Recursive calls
- No misleading or incorrect comments

4. Beginner-Level Understandability Assessment

✓ Strengths

- Simple variable names (`low`, `high`, `mid`)

- Clear separation of base and recursive cases
- Inline comments for every important step
- Easy-to-follow example

⚠ Minor Improvements (Optional)

- Could add a comment stating **list must be sorted**
- Could print intermediate steps for visualization (learning purpose)

5. Student's Assessment (Clarity, Correctness, Transparency)

- **Clarity:** ⭐ ⭐ ⭐ ⭐ ⭐ (Very clear for beginners)
- **Correctness:** ⭐ ⭐ ⭐ ⭐ ⭐ (Logic and implementation are correct)
- **Transparency:** ⭐ ⭐ ⭐ ⭐ ⭐ (Each step is explainable and traceable)

6. Conclusion

The AI-generated recursive binary search code is:

- Transparent
- Well-documented
- Beginner-friendly
- Easy to verify and debug

Task 5: Inclusiveness and Ethical Variable Design

Scenario

Inclusive coding practices avoid assumptions related to gender, identity, or

roles and promote fairness in software design.

Task Description

Use an AI tool to generate a Python code snippet that processes user or

employee details.

Analyze the code to identify:

• Gender-specific variables (e.g., male, female)

• Assumptions based on gender or identity

Non-inclusive naming or logic

Modify or regenerate the code to:

• Use gender-neutral variable names

• Avoid gender-based conditions unless strictly required

• Ensure inclusive and respectful coding practices

Expected Output

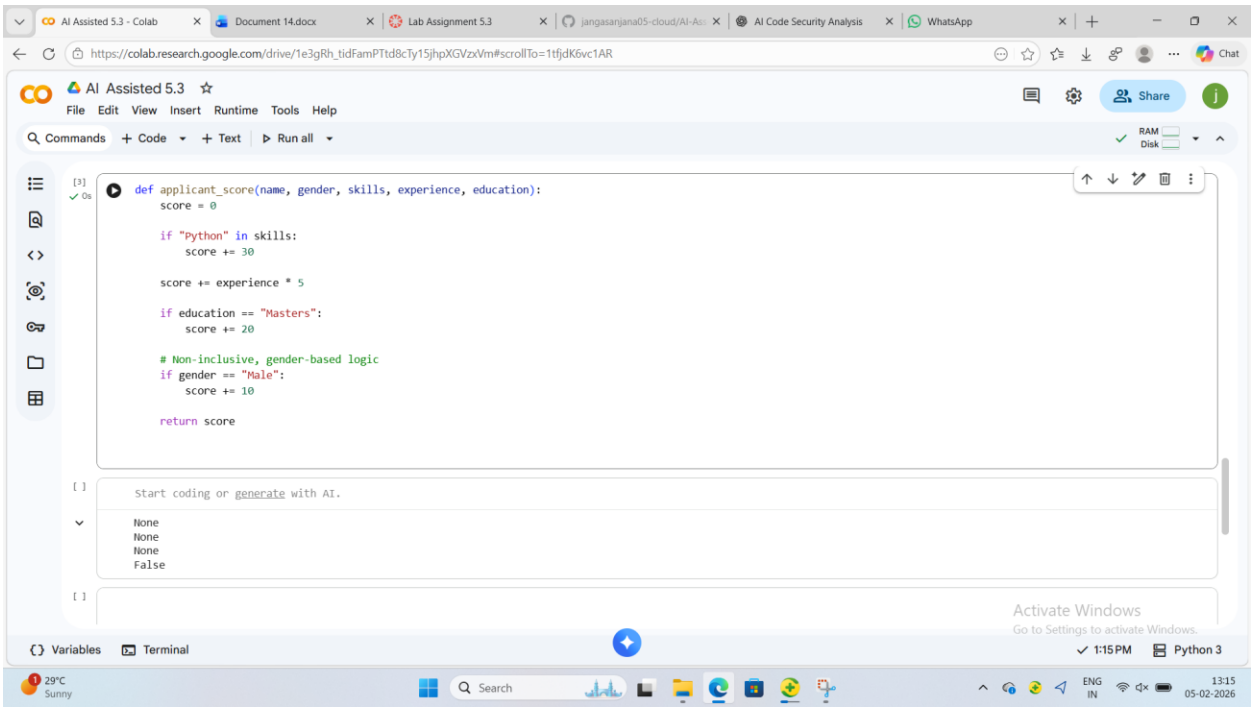
• Original AI-generated code snippet

• Revised inclusive and gender-neutral code

• Brief explanation of:

o What was non-inclusive

o How inclusiveness was improved



2. Issues with Non-Inclusive Naming and Logic

What Was Non-Inclusive

1. **Gender-based condition**
    a. Gives preference to one gender without job relevance.
2. **Use of sensitive personal attribute**
    a. gender is unrelated to skills, experience, or education.
3. **Unfair scoring**
    a. Identical candidates receive different scores due to gender.

4. How Inclusiveness Was Improved

Improvements Made

1. **Gender-neutral variables**
    a. Removed name and gender parameters entirely.
2. **Eliminated gender-based logic**
    a. Scoring depends only on job-relevant attributes.
3. **Fair and respectful design**
    a. Equal qualifications → equal scores.

4. **Inclusive coding practice**
    a. Avoids assumptions, stereotypes, or discriminatory logic.

5. Conclusion

The revised code:

- Uses **inclusive, gender-neutral logic**
- Ensures **fair evaluation**
- Aligns with **ethical AI and responsible coding standards**